

Efficient Decomposition Algorithm for Stationary Analysis of Complex Stochastic Petri Net Models

Kristóf Marussy¹, Attila Klenik¹, Vince Molnár², András Vörös²,
István Majzik¹, and Miklós Telek³

¹ Department of Measurement and Information Systems,
Budapest University of Technology and Economics,
Budapest, Hungary

² MTA-BME Lendület Cyber-Physical Systems Research Group,
Budapest, Hungary
`vori@mit.bme.hu`

³ MTA-BME Information Systems Research Group,
Budapest, Hungary

Abstract. Stochastic Petri nets are widely used for the modeling and analysis of non-functional properties of critical systems. The state space explosion problem often inhibits the numerical analysis of such models. Symbolic techniques exist to explore the discrete behavior of even complex models, while block Kronecker decomposition provides memory-efficient representation of the stochastic behavior. However, the combination of these techniques into a stochastic analysis approach is not straightforward. In this paper we integrate saturation-based symbolic techniques and decomposition-based stochastic analysis methods. Saturation-based exploration is used to build the state space representation and a new algorithm is introduced to efficiently build block Kronecker matrix representation to be used by the stochastic analysis algorithms. Measurements confirm that the presented combination of the two representations can expand the limits of previous approaches.

Keywords: Stochastic Petri nets · stationary analysis · block Kronecker decomposition · numerical algorithms · symbolic methods.

1 Introduction

Stochastic analysis provides information about the quantitative aspects of models and is used for the analysis of non-functional properties of critical systems. Stochastic Petri nets are widely used in reliability, availability and performance modelling to capture the stochastic behaviours and the analysis questions are answered with the help of Markovian analysis. However, successful stochastic analysis is often prevented by the *state space explosion problem*: in addition to the complexity of traditional qualitative analysis, stochastic computations require more involved data structures and numerical algorithms. To successfully tackle these problems, efficient algorithms are needed.

Many efficient techniques exist in the literature for the exploration and storage of the state space of Petri net models. One of the most efficient is the so-called saturation algorithm which uses a special iteration strategy for the state space traversal and stores the state space representation symbolically in decision diagrams [13]. On the other hand, numerical algorithms for Markovian analysis necessitates the representation of the stochastic behaviours.

The infinitesimal generator matrix describes the behaviours of stochastic Petri nets and the underlying Markov chains. The size of the matrix is quadratic in the number of reachable states of the system. This implies a quadratic storage complexity in case the matrix is directly represented in a dense form. Sparse matrix formats, such as Compressed Column Storage (CCS) [3, Section 4.3.1], reduce memory requirements to be proportional to the transitions in the system. However, even sparse storage techniques tend to fail quickly due to state space explosion.

Potential Kronecker methods [8] divide the large matrix representation into smaller matrices using only local information. Computations are then performed with the local matrices and vectors. Unfortunately, using local information leads to storing probabilities for unreachable states. This may cause problems in some numerical solver algorithms as well as increases storage requirements.

In contrast, *actual Kronecker methods* [5, 8, 18] apply additional conversions and computations to handle unreachable states in the encoding. This yields higher implementation complexities and computational overhead.

The basis of our work is *block Kronecker decomposition* which imposes a hierarchical structure on the reachable state space to solve the issue of unreachable potential states [4, 7, 9].

Several algorithms have been developed that use variations of decision diagrams to represent the infinitesimal generator. Matrix diagrams [12, 21] generalize the Kronecker representation to arbitrary matrices and not necessarily Kronecker consistent model partitions. Multi-Terminal Decision Diagrams (MTDDs) can store both the generator matrix [20] and the vector of state probabilities [19] by extending decision diagrams with terminal nodes corresponding to real numbers. Multiplicative Edge-valued Multilabel Decision Diagrams (EV*MDDs) [25] can provide up to exponential space savings compared to MTDDs by storing matrix and vector entries as edge labels instead of terminal nodes.

While vector-matrix products can be handled with symbolic approaches easily, more elaborate matrix access becomes difficult. For example, efficient access of a single column of the symbolic descriptor requires the introduction of caching strategies [26].

As it turns out from the literature, the combination of efficient state space traversal and matrix representation techniques into a stochastic analysis approach is not straightforward. In this paper we elaborate an idea of Buchholz [10] to construct the matrix representation of the stochastic behaviour from symbolic state space representation. In addition to the initial idea of Buchholz [10], we further extended the method by employing partition refinement instead of hashing and we also proved correctness of the algorithm formally. The stochastic analysis

framework uses saturation-based exploration to build the state space representation and the new algorithm builds the block Kronecker matrix representation to be used by the stochastic analysis algorithms. Measurements confirm that the presented combination of the two representations can expand the limits of previous approaches.

2 Background

In this section, we overview the basic formalisms and scope of our work. At first, a stochastic Petri net based formalism is introduced. Kronecker algebra and multivalued decision diagrams are also discussed.

2.1 Stochastic Petri Nets

Stochastic Petri Nets extend Petri nets by assigning exponentially distributed random delays to transitions [1]. After the delay associated with an enabled transition is elapsed the transition fires and transition delays are reset.

Definition 1. *A Stochastic Petri Net is a pair $SPN = \langle PN, \Lambda \rangle$, where PN is a Petri net and $\Lambda: T \rightarrow \mathbb{R}^+$ maps the set of transitions to transition rates. PN may contain inhibitor arcs but no priority specifications.*

The stochastic behaviours of a stochastic Petri net are defined by an underlying continuous-time Markov chain (CTMC). The Markov chain associated with an SPN is a stochastic process $X(\tau) \in \text{RS}$, $\tau \geq 0$, where RS is the set of reachable markings of the underlying Petri net.

We will only consider the case when the Petri net is bounded, hence $n = |\text{RS}| < \infty$. In order to establish the transformation between the Petri net and its underlying Markov chain, we have to define a mapping from states to indices. A bijection $\iota: \text{RS} \rightarrow \{0, 1, \dots, n-1\}$ exists between the reachable markings and a set of indices. This allows representing the distribution of $X(\tau)$ as a vector

$$\boldsymbol{\pi}(\tau) \in \mathbb{R}^n, \quad \pi(\tau)[x] = \Pr(X(\tau) = \iota^{-1}(x)),$$

i.e. $\pi(\tau)[\iota(M)]$ is the probability that the SPN is in the marking M at time τ .

The time evolution of the $X(\tau)$ is described by the differential equation

$$\frac{\partial \boldsymbol{\pi}(\tau)}{\partial \tau} = \boldsymbol{\pi}(\tau) Q, \tag{1}$$

where $Q \in \mathbb{R}^{n \times n}$ is the *infinitesimal generator matrix* of the CTMC associated with the stochastic Petri net.

Off-diagonal elements $q[x, y]$ of Q ($0 \leq x, y < n$ and $x \neq y$) contain the rate of exponentially distributed transitions from the marking $\iota^{-1}(x)$ to $\iota^{-1}(y)$. Diagonal elements $q[x, x]$ are calculated such that $Q\mathbf{1}^T = \mathbf{0}^T$, where $\mathbf{1}$ and $\mathbf{0} \in \mathbb{R}^n$ are vectors with every element equal to 1 and 0, respectively. That is,

$$q[x, y] = \begin{cases} -\sum_{z=0, z \neq x}^{n-1} q[x, z] & \text{if } x = y, \\ \sum_{t \in T} \sum_{\iota^{-1}(x)[t] \iota^{-1}(y)} \Lambda(t) & \text{if } x \neq y. \end{cases} \tag{2}$$

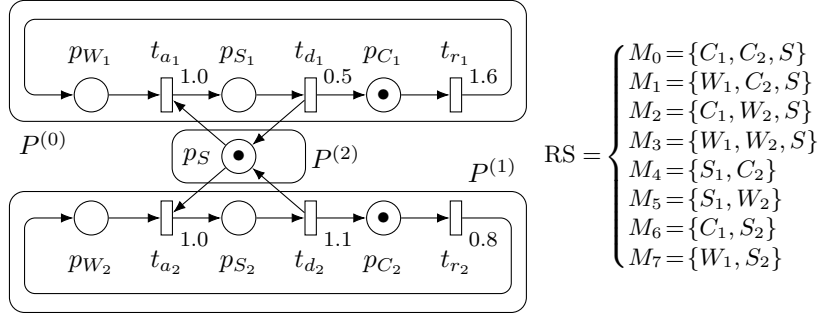


Fig. 1. Example stochastic Petri net with superposed partitions.

The notation $\iota^{-1}(x) [t] \iota^{-1}(y)$ indicates that the transition t can be fired in the marking $\iota^{-1}(x)$ to assume the marking $\iota^{-1}(y)$.

Example 1. In Fig. 1 we introduce the *SharedResource* model which will serve as a running example throughout this paper.

The model consists of a single shared resource S and two consumers. Each consumer can be in one of the following states: C_i (calculating locally), W_i (waiting for resource) and S_i (using shared resource). The transitions r_i (request resource), a_i (acquire resource) and d_i (done) correspond to behaviors of the consumers. The net has 8 reachable states, which are also shown in Fig. 1. As the net is 1-bounded, only the marked places are listed for each state.

The net is annotated with exponentially distributed transition rates. The clients have different request (1.6 and 0.8) and completion (0.5 and 1.1, respectively) rates, while both clients have an acquire rate of 1.0.

2.2 Superposed Stochastic Petri Nets

As the decomposition method in our approach fits the concept of superposed Petri nets, we use it in the rest of the paper.

Definition 2. A Superposed Stochastic Petri Net (SSPN) is a pair $SSPN = \langle SPN, \mathcal{P} \rangle$, where $\mathcal{P} = \{P^{(0)}, P^{(1)}, \dots, P^{(J-1)}\}$ is the partitioning of the set of places P in the underlying Petri net of SPN such that $P = P^{(0)} \sqcup \dots \sqcup P^{(J-1)}$ [15].

The partition $P^{(j)}$ is called the j th local net or component of SSPN. A local marking $M^{(j)}: P^{(j)} \rightarrow \mathbb{N}$ is obtained from a global marking $M: P \rightarrow \mathbb{N}$ by restricting the domain to $P^{(j)}$, i.e. $M^{(j)} = M|_{P^{(j)}}$.

The local reachable state space of the j th local net contains the restrictions of the globally reachable markings $RS^{(j)} = \{M^{(j)} : M \in RS\}$.

We will assume a bijection $\iota^{(j)}: RS^{(j)} \rightarrow \{0, 1, \dots, n_j\}$ from local markings to an index set, where $n_j = |RS^{(j)}|$. Let the notation $x^{(j)}$ refer to the local marking $(\iota^{(j)})^{-1}(x)$.

If $\mathbf{x} = \langle x[0], x[1], \dots, x[J-1] \rangle$ is a vector of indices, then let the notation

$$M = \langle\langle \mathbf{x} \rangle\rangle = \langle\langle (x[0])^{(0)}, (x[1])^{(1)}, \dots, (x[J-1])^{(J-1)} \rangle\rangle$$

refer to the marking M with the property $M^{(j)} = (x[j])^{(j)}$ for all j , i.e. M is the marking obtained by joining the local markings indexed by \mathbf{x} . We extend this notation to take sets of local states and yield a set of markings.

The *potential state space*

$$\text{PS} = \langle\langle \text{RS}^{(0)}, \text{RS}^{(1)}, \dots, \text{RS}^{(J-1)} \rangle\rangle$$

is isomorphic to the Cartesian product of local state spaces. More concretely, each $M \in \text{PS}$ can be identified with a vector of indices \mathbf{x} such that $M = \langle\langle \mathbf{x} \rangle\rangle$, i.e. M can be obtained by joining some local states for each component. This vector is the state coding of M expressed by the function $\iota(M) : \text{PS} \rightarrow \mathbb{N}^J$, $\iota(\langle\langle \mathbf{x} \rangle\rangle) = \mathbf{x}$.

Let us write $x^{(j)} [t] y^{(j)}$ if there is a reachable marking $M_x \in \text{RS}$ such that

$$M_x|_{P^{(j)}} = x^{(j)}, \quad M_x [t] M_y, \quad M_y|_{P^{(j)}} = y^{(j)},$$

i.e. there is a global state transition that takes the j th local net from the state $x^{(j)}$ to $y^{(j)}$. It is important to note that superposed stochastic Petri nets are *Kronecker consistent*: if $x^{(j)} [t] y^{(j)}$ and $x^{(j)} [t] z^{(j)}$, then $y^{(j)} = z^{(j)}$.

Example 2. The *SharedResource* SPN in Fig. 1 contains three partitions. The local nets $P^{(0)}$ and $P^{(1)}$ correspond to the two clients, while $P^{(2)}$ contains the shared resource. The local state spaces of the components are

$$\text{RS}^{(0)} = \begin{cases} 0^{(0)} = \{C_1\} \\ 1^{(0)} = \{W_1\} \\ 2^{(0)} = \{S_1\} \end{cases}, \quad \text{RS}^{(1)} = \begin{cases} 0^{(1)} = \{C_2\} \\ 1^{(1)} = \{W_2\} \\ 2^{(1)} = \{S_2\} \end{cases}, \quad \text{RS}^{(2)} = \begin{cases} 0^{(2)} = \{S\} \\ 1^{(2)} = \emptyset \end{cases}.$$

The reachable states can be factored over the local state spaces, e.g. the marking $M_4 = \{S_1, C_2\} = \langle\langle 2^{(0)}, 0^{(1)}, 1^{(2)} \rangle\rangle$. However, the potential state space PS contains $|\text{RS}^{(0)}| \cdot |\text{RS}^{(1)}| \cdot |\text{RS}^{(2)}| = 3 \cdot 3 \cdot 2 = 18$ states, 10 more than the reachable state space RS. For example, the marking $\langle\langle 2^{(0)}, 2^{(1)}, 0^{(2)} \rangle\rangle = \{S_1, S_2, S\}$, which violates mutual exclusion, is not reachable, although it is in PS.

2.3 Decision Diagrams

Multivalued decision diagrams (MDDs) [13] provide a compact, graph-based representation for boolean functions defined over Cartesian products of domains.

Definition 3. A *quasi-reduced ordered multivalued decision diagram (MDD)* encoding the function $f(x[0], x[1], \dots, x[J-1]) \in \{0, 1\}$, where the domain of each variable $x[j]$ is $D^{(j)} = \{0, 1, \dots, n_j - 1\}$, is a tuple $\text{MDD} = \langle V, r, \underline{0}, \underline{1}, \text{level}, \text{child} \rangle$, where

- $V = \bigcup_{i=0}^J V_i$ is a finite set of nodes, where $V_0 = \{\underline{0}, \underline{1}\}$ are the terminal nodes, the rest of the nodes $V_N = V \setminus V_0$ are nonterminal nodes;
- $level: V \rightarrow \{0, 1, \dots, J\}$ assigns nonnegative level numbers to each node, i.e. $V_i = \{\underline{v} \in V : level(\underline{v}) = i\}$;
- $\underline{r} \in V_J$ is the root node;
- $\underline{0}, \underline{1} \in V_0$ are the zero and one terminal nodes;
- $child: (\bigcup_{i=1}^J V_i \times D^{(i-1)}) \rightarrow V$ is a function defining edges between nodes labeled by the items of the domains, such that either $child(\underline{v}, x) = \underline{0}$ or $level(child(\underline{v}, x)) = level(\underline{v}) - 1$ for all $\underline{v} \in V$, $x \in D^{(level(\underline{v})-1)}$;
- if $\underline{n}, \underline{m} \in V_j, j > 0$ then the subgraphs formed by the nodes reachable from \underline{n} and \underline{m} are either non-isomorphic, or $\underline{n} = \underline{m}$.

According to the semantics of MDDs, $f(\mathbf{x}) = 1$ if the node $\underline{1}$ is reachable from \underline{r} through the edges labeled with $x[0], x[1], \dots, x[J-1]$,

$$f(x[0], x[1], \dots, x[J-1]) = 1 \iff child(child(\dots child(\underline{r}, x[J-1]) \dots, x[1]), x[0]) = \underline{1}.$$

Definition 4. A quasi-reduced ordered edge-valued multivalued decision diagram (EDD) [22] encoding the function $g(x^{(0)}, x^{(1)}, \dots, x^{(J-1)}) \in \mathbb{N}$ is a tuple $EDD = \langle V, \underline{r}, \underline{0}, \underline{1}, level, child, label \rangle$, where

- $MDD = (V, \underline{r}, \underline{0}, \underline{1}, level, child)$ is a quasi-reduced ordered MDD;
- $label: (\bigcup_{i=1}^J V_i \times D^{(i-1)}) \rightarrow \mathbb{N}$ is an edge label function.

According to the semantics of EDDs, the function g is evaluated by summing edge labels along the path from \underline{r} to $\underline{1}$. Formally,

$$g(\mathbf{x}) = \begin{cases} \text{undefined} & \text{if } f(\mathbf{x}) = 0, \\ \sum_{j=0}^{J-1} label(\underline{n}^{(j)}, x[j]) & \text{if } f(\mathbf{x}) = 1, \end{cases}$$

where f is the function associated with the underlying MDD and $\underline{n}^{(j)}$ are the nodes along the path to $\underline{1}$, i.e. $\underline{n}^{(J-1)} = \underline{r}$, $\underline{n}^{(j)} = child(\underline{n}^{(j+1)}, x[j+1])$.

Symbolic State Spaces. Symbolic techniques involving MDDs can efficiently store large reachable state spaces of superposed Petri nets. Reachable states $M \in \text{RS}$ are associated with state codings $\iota(M) = \mathbf{x}$. The function $f: \text{PS} \rightarrow \{0, 1\}$ can be stored as an MDD where $f(\mathbf{x}) = 1$ if and only if $\langle \mathbf{x} \rangle \in \text{RS}$. The domains of the MDD are the local state spaces $D^{(j)} = \text{RS}^{(j)}$.

Similarly, EDDs can efficiently store the mapping between symbolic state encodings \mathbf{x} and reachable state indices $x = \iota(\langle \mathbf{x} \rangle)$ as the function $g(\mathbf{x}) = x$. This mapping is used to refer to elements of state probability vectors and the sparse generator matrix Q when these objects are created and accessed [12].

Some iteration strategies for MDD state space exploration are *breath-first search* and *saturation* [13].

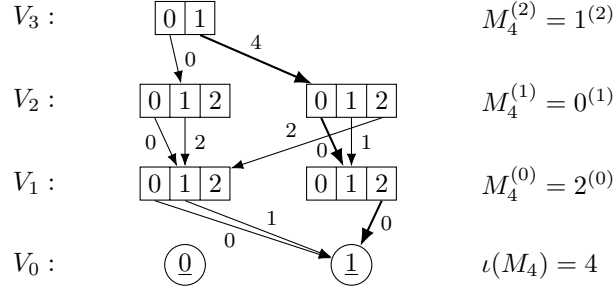


Fig. 2. EDD state space mapping of the *SharedResource* model.

Example 3. The EDD displayed in Fig. 2 describes the reachable state space of the *SharedResource* model from Example 2. Edges to $\underline{0}$ were omitted for clarity.

The edge labels allow computation of the indexing function ι for a given state. For example, to find the index of the marking $M_4 = \langle\langle 2^{(0)}, 0^{(1)}, 1^{(2)} \rangle\rangle$, we can follow the edges corresponding to the local states to $\underline{1}$ and sum their labels to find $\iota(M_4) = 4 + 0 + 0 = 4$. In contrast, if we follow the edges corresponding to the unreachable state $\langle\langle 2^{(0)}, 2^{(1)}, 0^{(2)} \rangle\rangle$, $\underline{0}$ is reached instead of $\underline{1}$.

2.4 Kronecker Algebra

In linear algebra, the Kronecker product operation may be used to build large matrices from smaller ones. It therefore plays an important role in the stochastic analysis of Markovian systems, where Kronecker products of matrices may help reducing the memory requirements of the infinitesimal generator matrix Q .

Definition 5. The Kronecker product $A \otimes B$ of matrices $A \in \mathbb{R}^{n_1 \times m_1}$ and $B \in \mathbb{R}^{n_2 \times m_2}$ is the matrix $C \in \mathbb{R}^{n_1 n_2 \times m_1 m_2}$, where

$$c[i_1 n_1 + i_2, j_1 m_1 + j_2] = a[i_1, j_1] b[i_2, j_2].$$

The SHUFFLE family of algorithms [5] allows efficient evaluation of vector-matrix products of the form

$$\mathbf{v} \cdot (A^{(0)} \otimes A^{(1)} \otimes \dots \otimes A^{(J-1)}).$$

The factors $A^{(j)} \in \mathbb{R}^{n_j \times m_j}$ together represent an $n_0 n_1 \dots n_{J-1} \times m_0 m_1 \dots m_{J-1}$ matrix in the J -way Kronecker product.

Recent developments include the SLICE [16] and SPLIT [14] algorithms for vector-descriptor products, which allow parallel implementation while retaining the beneficial properties of the SHUFFLE algorithm.

3 Stochastic Petri Net Stationary Analysis

In this section the used stochastic analysis approach is introduced.

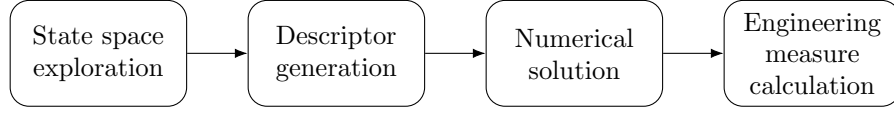


Fig. 3. Stochastic analysis workflow.

3.1 Analysis Workflow

The tasks performed by stochastic analysis tools that operate on stochastic Petri nets, can be often structured as follows (Fig. 3).

1. *State space exploration.* The reachable state space of the Petri net is explored to enumerate the possible behaviors of the model. For superposed stochastic Petri nets, this step includes the exploration of the local state spaces of the component as well as the possible global combinations of states.
2. *Descriptor generation.* The *infinitesimal generator matrix* Q is built in order to describe the Markov chain $X(t)$ over the reachable states of the stochastic Petri net.
3. *Numerical solution.* Numerical algorithms obtain probability vectors π from the matrix Q .
4. *Engineering measure calculation.* The studied performance measures are calculated from the output of the previous step. The expected values of most measures of interest can be obtained as weighted sums of state probabilities.

In stochastic model checking, where the desired system behaviors are expressed in stochastic temporal logics [2, 6], these analytic steps are called as subroutines to evaluate propositions.

In the steady-state analysis of continuous-time Markovian stochastic systems, the steady state solution

$$\pi(0) = \pi_0, \quad \frac{\partial \pi(\tau)}{\partial \tau} = \pi(\tau) Q, \quad \pi = \lim_{\tau \rightarrow \infty} \pi(\tau) \quad (3)$$

of Eq. 1 is sought, where π_0 describes the initial probability distribution and π is the stationary solution. If the CMTC is irreducible, i.e. there is a nonzero probability of transitioning from any state to any other, π is independent from π_0 and is the initial solution of the system of linear equations

$$\pi Q = \mathbf{0}, \quad \pi \mathbf{1}^T = 1. \quad (4)$$

Example 4. The utilization of the shared resource in the *SharedResource* SPN, presented in Fig. 1, can be calculated as the sum

$$U = \pi[\iota(M_4)] + \pi[\iota(M_5)] + \pi[\iota(M_6)] + \pi[\iota(M_7)]$$

after obtaining π from Eq. (4). Notice that the resource is in use in the reachable markings M_4 , M_5 , M_6 and M_7 .

To solve Eq. (4), the matrix Q and the vector π must be stored. Additionally, the numerical algorithm may reserve additional vectors for intermediate storage.

3.2 Infinitesimal Generator Matrix Storage

In this section the basic complexity issues behind our developments are summarized. Traditional methods use sparse or dense matrix storage methods.

Dense and Sparse Matrices. The infinitesimal generator matrix Q of a stochastic Petri net is an $n \times n$ matrix of real numbers, where n is the number of reachable states. The storage of Q thus requires memory proportional to the square of the state space size if a dense matrix form is used.

Sparse matrix representation [3, Section 4.3.1], reduces memory requirements to $O(\text{NZ})$, where NZ is the number of nonzero elements in Q .

Kronecker Decomposition. To alleviate the high memory requirements of Q , the Kronecker decomposition for a superposed SPN expresses the infinitesimal generator matrix as a sum of Kronecker products. Let

$$Q = Q_O + Q_D, \quad Q_D = \text{diag}\{-Q\mathbf{1}^T\}, \quad (5)$$

where Q_O and Q_D are the off-diagonal and diagonal parts of Q , respectively. The off-diagonal part may be written as

$$Q_O = \sum_{t \in T} \Lambda(t) \bigotimes_{j=0}^{J-1} Q_t^{(j)}. \quad (6)$$

The matrix $Q_t^{(j)} \in \mathbb{R}^{n_j \times n_j}$ describes the effects of the transition t on the j th local net. If $t \in p^\bullet \cup \bullet p \cup p^\circ$ for some $p \in P^{(j)}$, i.e. t is connected to a place in the j th local net with an input, output or inhibitor arc,

$$q_t^{(j)}[x, y] = \begin{cases} 1 & \text{if } x^{(j)} [t] y^{(j)}, \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

If t is not adjacent to the j th local net, $Q_t^{(j)}$ is set to an $n_j \times n_j$ identity matrix.

3.3 Block Kronecker Decomposition

In this section, we review the concept of block Kronecker decomposition and hierarchical structuring of the state space from [7], which divides big monolithic matrices into smaller pieces by the use of *macro states*.

Let $\widetilde{\text{RS}}^{(j)}$ denote the *local macro states* of the j th local net of an SSPN. Elements of $\widetilde{\text{RS}}^{(j)}$ form a partition of the local state space $\text{RS}^{(j)}$ of the j th component, i.e. $\text{RS}^{(j)} = \bigsqcup \widetilde{\text{RS}}^{(j)}$.

The macro state indexing function $\tilde{i}^{(j)}: \widetilde{\text{RS}}^{(j)} \rightarrow \{0, 1, \dots, \tilde{n}_j - 1\}$ assigns a unique index to every macro state, where $\tilde{n}_j = |\widetilde{\text{RS}}^{(j)}|$. We use the notation $\tilde{m}^{(j)}$ to refer to $(\tilde{i}^{(j)})^{-1}(m) \subseteq \text{RS}^{(j)}$.

A vector $\mathbf{m} = \langle m[0], m[1], \dots, m[J-1] \rangle$ is a *global macro state index* if

$$\langle \widetilde{m[0]}^{(0)}, \widetilde{m[1]}^{(1)}, \dots, \widetilde{m[J-1]}^{(J-1)} \rangle \subseteq \text{RS},$$

i.e. a subset of reachable markings is isomorphic to a Cartesian product of local macro states. Such subset is called a *global macro state*.

Hierarchical structuring of the reachable state space expresses RS as a disjoint union of global macro states $\widetilde{\text{RS}} = \{\tilde{0}, \tilde{1}, \dots, \tilde{n} - 1\}$.

The bijection $\tilde{\iota}: \widetilde{\text{RS}} \rightarrow \{0, 1, \dots, \tilde{n} - 1\}$ assigns indices to global macro states such that $\tilde{\iota}^{-1}(m) = \tilde{m}$, while the function $\tilde{\iota}: \widetilde{\text{RS}} \rightarrow \mathbb{N}^J$ assigns their respective global macro state index vectors.

Since each global macro state has the structure of a Cartesian product of local macro states, the Kronecker product may be used to construct a matrix of state transitions between two global macro states.

Let \mathbf{m} and \mathbf{k} be two reachable global macro state indices. The matrix $Q_t^{(j)}[m[j], k[j]]$ is obtained from $Q_t^{(j)}$ in Eq. (7) by only keeping the rows that correspond to $m[j]^{(j)}$ and the columns that correspond to $k[j]^{(j)}$. The matrix

$$Q_O[m, k] = \sum_{t \in T} \Lambda(t) \bigotimes_{j=0}^{J-1} Q_t^{(j)}[m[j], k[j]] \quad (8)$$

describes the states transitions from the macro state $\tilde{\iota}^{-1}(m)$ to $\tilde{\iota}^{-1}(k)$, where m and k are the indices of global macro states with index vectors \mathbf{m} and \mathbf{k} , respectively.

We can finally express the infinitesimal generator matrix Q as a block matrix with $\tilde{n} \times \tilde{n}$ blocks. Let the $\langle m, k \rangle$ th block of the off-diagonal part Q_O be $Q_O[m, k]$ as defined in Eq. (8) above. Then it can be seen that the matrix

$$Q = Q_O + Q_D, \quad Q_D = \text{diag}\{-Q\mathbf{1}^T\} \quad (9)$$

is equivalent to the matrix Q in Eq. (2).

Example 5. The state space of the *SharedResource* model in Fig. 1 may be hierarchically structured as follows.

Recall from Example 2 that the model has three local nets, two corresponding to the clients with three local states each and a local net with two local states corresponding to the shared resource. We may partition the local state spaces into local macro states as

$$\widetilde{RS}^{(0)} = \left\{ \begin{array}{l} \tilde{0}^{(0)} = \{0^{(0)}, 1^{(0)}\} \\ \tilde{1}^{(0)} = \{2^{(0)}\} \end{array} \right\}, \quad \widetilde{RS}^{(1)} = \left\{ \begin{array}{l} \tilde{0}^{(1)} = \{0^{(1)}, 1^{(1)}\} \\ \tilde{1}^{(1)} = \{2^{(1)}\} \end{array} \right\}, \quad \widetilde{RS}^{(2)} = \left\{ \begin{array}{l} \tilde{0}^{(2)} = \{0^{(2)}\} \\ \tilde{1}^{(2)} = \{1^{(2)}\} \end{array} \right\}.$$

Observe that, for each component j , the macro state $\tilde{0}^{(j)}$ contains local states that are reachable when the shared resource is not in use, while $\tilde{1}^{(j)}$ corresponds to the allocation of the resource.

The global reachable state space is then partitioned into global macro states as

$$\widetilde{\text{RS}} = \{\langle\langle\tilde{o}^{(0)}, \tilde{o}^{(1)}, \tilde{o}^{(2)}\rangle\rangle, \langle\langle\tilde{o}^{(0)}, \tilde{i}^{(1)}, \tilde{i}^{(2)}\rangle\rangle, \langle\langle\tilde{i}^{(0)}, \tilde{o}^{(1)}, \tilde{i}^{(2)}\rangle\rangle\},$$

i.e. any state that does not require use of the shared resource is reachable when the shared resource is available, while if one of the clients allocates the resource, the other cannot simultaneously acquire it.

Macro State Construction. Let us introduce the notation

$$\hat{\mathbf{x}}^{(j)} = \langle x[0], x[1], \dots, x[j-1], x[j+1], \dots, x[J-1] \rangle.$$

Suppose that the reachable marking M is coded by the vector of local state indices $\mathbf{x} = \boldsymbol{\iota}(M)$, i.e. $M = \langle\langle\mathbf{x}\rangle\rangle$. Then the vector $\hat{\mathbf{x}}^{(j)}$ contains the local state indices of all components except the j th.

The *environment* of a local state $x^{(j)}$ is the set of vectors

$$\text{env } x^{(j)} = \{\hat{\mathbf{z}}^{(j)} : M \in \text{RS}, \mathbf{z} = \boldsymbol{\iota}(M), z[j] = x\}, \quad (10)$$

i.e. the local state combinations of other local nets that result in reachable markings together with $x^{(j)}$. We define the equivalence relation $\sim^{(j)} \subseteq \text{RS}^{(j)} \times \text{RS}^{(j)}$,

$$x^{(j)} \sim^{(j)} y^{(j)} \iff \text{env } x^{(j)} = \text{env } y^{(j)}. \quad (11)$$

The equivalence classes $\text{RS}^{(j)} / \sim^{(j)}$ define the local macro states $\widetilde{\text{RS}}^{(j)}$. Now we can construct the equivalence relation $\sim \subseteq \text{RS} \times \text{RS}$,

$$M \sim K \iff M^{(j)} \sim^{(j)} K^{(j)} \text{ for all } j = 0, 1, \dots, J-1,$$

i.e. two markings are equivalent if all their local nets belong to the same local macro states. The set of global macro states is the partition $\widetilde{\text{RS}} = \text{RS} / \sim$.

Explicit Macro State Algorithm. The original hierarchical structuring algorithm proposed by Buchholz is based on a bit array representation of the potential state space PS [7].

Algorithm 1 shows local and global macro state generation. The environment $\text{env } x^{(j)}$ is represented explicitly as a row of the reshaped bit array \mathbf{B} . Nonzero elements correspond to reachable states.

Lexicographic ordering is used to make local states with equal environments adjacent. After extracting local macro states from the bit array, only a single representant of every local macro state is kept. This both accelerates further iterations of the algorithm and results in reduced bit array at the end that has a nonzero element for every combination of local states that is reachable. Note that the permutations used to reorder local states must be stored in order to recover the global macro state indices.

Note that storage of the bit array requires $O(|\text{PS}|)$ memory, therefore the bit array based macro state generation is unsuitable for extremely large potential state spaces due to memory requirements.

Algorithm 1: Bit array based macro state construction.

Input: Reachable state space RS , reachable local states $RS^{(j)}$
Output: Local macro states $\widetilde{RS}^{(j)}$, global macro states \widetilde{RS}

- 1 Allocate an array of bits $\mathbf{B} \in \{0, 1\}^{n_0 \times n_1 \times \dots \times n_{J-1}}$
- 2 **foreach** $M \in RS$ **do** $\mathbf{x} \leftarrow \iota(M)$; $B[x[0], x[1], \dots, x[J-1]] \leftarrow 1$
- 3 **for** $j \leftarrow 0$ **to** $J-1$ **do**
- 4 Reshape \mathbf{B} into a matrix with n_j , where the x th row corresponds to the local state $x^{(j)} \in RS^{(j)}$ and its environment $\text{env } x^{(j)}$
- 5 Sort the rows of the matrix lexicographically
- 6 Partition the rows such that equal rows form local macro states $\widetilde{RS}^{(j)}$
- 7 Discard all but one representant row for each local macro state $\widetilde{m}^{(j)}$
- 8 Nonzero elements of the resulting bit array \mathbf{B} are correspond to the reachable global macro states \widetilde{RS}

4 Symbolic Decomposition Algorithm

In this section, we present our symbolic decomposition algorithm that allows the construction of macro state spaces without explicit enumeration and storage of the potential state space PS.

4.1 Description

The memory requirements and run time of bit array based macro state decomposition may be significantly improved by the use of symbolic state space storage instead of a bit vector. Algorithm 2 constructs the local macro states from the MDD representation of the state space.

The algorithm partitions the local states $RS^{(j)}$ of every component of the stochastic Petri net into local macro states. In order to perform this operation, the nodes in the levels V_1, V_2, \dots, V_J of the MDD corresponding to the state space RS must be provided. While symbolic techniques often share nodes between multiple MDDs, in Algorithm 2 the sets V_j should contain only nodes reachable from the root \underline{r} corresponding to RS . These sets can be extracted from a shared MDD container by e.g. depth first search starting at \underline{r} .

The lines 2–13 implement partition refinement for $RS^{(j)}$ based on the local states associated with the edges between nodes in V_{j+1} and V_j , i.e. the j th level of the MDD.

The candidate macro state partition $Done$ is initialized to contain only $RS^{(j)}$. Then, the lines 5–12 refine the candidate macro states for each node $\underline{n} \in V_{j+1}$ according to the associated local states of the arcs starting at \underline{n} . After moving the sets from $Done$ to the queue Q , every candidate macro state $S \in Q$ is split into S_1 and S_2 . Arcs from \underline{n} with local states $x \in S_1$ all go to some node $\underline{m} \in V_j$. The candidate macro state S_1 is added to the new $Done$ partition. Arcs from \underline{n} with local states $y \in S_2$ may not be all parallel, therefore S_2 is placed back to Q if it is nonempty.

Algorithm 2: Local macro state construction by partition refinement.

Input: Symbolic state space MDD
Output: Local macro states $\widetilde{\text{RS}}^{(j)}$

```

1 for  $j \leftarrow 0$  to  $J - 1$  do
2   Initialize the empty queue  $Q$  and  $Done \leftarrow \{\text{RS}^{(j)}\}$ 
3   foreach  $\underline{n} \in V_{j+1}$  do
4     foreach  $S \in Done$  do ENQUEUE( $Q, S$ )
5      $Done \leftarrow \emptyset$ 
6     while  $\neg \text{EMPTY}(Q)$  do
7        $S \leftarrow \text{DEQUEUE}(Q)$ ;  $S_1 \leftarrow \emptyset$ ;  $S_2 \leftarrow \emptyset$ 
8       Let  $x_0$  be any element of  $S$  and  $\underline{m} \leftarrow \text{child}(\underline{n}, x_0)$ 
9       foreach  $x \in S \setminus \{x_0\}$  do
10        if  $\underline{m} = \text{child}(\underline{n}, x)$  then  $S_1 \leftarrow S_1 \cup \{x\}$  else  $S_2 \leftarrow S_2 \cup \{x\}$ 
11        if  $S_2 \neq \emptyset$  then ENQUEUE( $Q, S_2$ )
12         $Done \leftarrow Done \cup \{S_1\}$ 
13    $\tilde{n}_j \leftarrow |Done|$ ;  $\widetilde{\text{RS}}^{(j)} \leftarrow Done$ 

```

After Q becomes empty and no new S_2 is enqueued, edges starting at \underline{n} which correspond to $x \in S$ all go to the same $\underline{m}_S \in V_j$ for all candidate macro states $S \in Done$. No two candidate macro states $S \neq S'$ have $\underline{m}_S = \underline{m}_{S'}$, due to the construction of the partition refinement. Moreover, this property also holds for nodes $\underline{n}' \in V_{j+1}$ processed in lines 5–12 before \underline{n} . Hence macro states in the final partition $Done$ constructed in lines 2–13 are all parallel in the sense that $\text{child}(\underline{n}, x) = \text{child}(\underline{n}, y)$ for all $\underline{n} \in V_{j+1}$ and $x, y \in S \in Done$.

In Subsection 4.2 we will prove that the final partition $Done$ is indeed the set of local macro states $\widetilde{\text{RS}}^{(j)}$.

Algorithm 2, unlike Algorithm 1, does not output the global macro state space $\widetilde{\text{RS}}$. An MDD representation of $\widetilde{\text{RS}}$ may be obtained by replacing every arc x with the index m of the local macro state such that $x^{(j)} \in \tilde{m}^{(j)}$. The original state space MDD can be recovered by the opposite operation, which replaces m with parallel arcs x_0, x_1, \dots having $\tilde{m}^{(j)} = \{x_0^{(j)}, x_1^{(j)}, \dots\}$.

Example 6. Fig. 4 shows the MDD state space of the *SharedResource* model, its local macro state decomposition and the MDD representation of the global macro states.

The edges corresponding to local states that belong to the same local macro state are parallel for *all* parent nodes on a level. This is represented in the figure with ellipses connecting the edges. In the global macro state MDD, these parallel edge sets collapse to a single edge, as every edge from the original MDD is replaced with its macro state.

Block Kronecked decomposition of the infinitesimal generator Q can be performed based on the sets of macro states $\widetilde{\text{RS}}^{(j)}$ obtained by the partition refinement. By enumerating paths in the macro state MDD, global macro states $\widetilde{\text{RS}}$

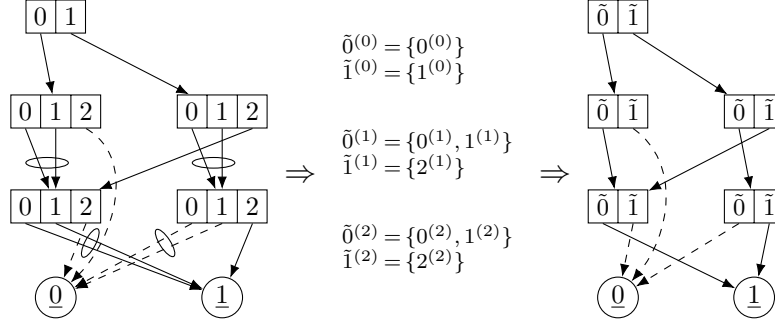


Fig. 4. Symbolic macro state construction for the *SharedResource* model.

can be listed. Moreover, the macro state MDD can be turned into an EDD for mapping indices between block Kronecker and sparse representations, which is described in Subsection 4.3.

4.2 Proof of Correctness

To show the correctness of the partition of the local states $RS^{(j)}$ into macro states $\tilde{RS}^{(j)}$, we will use the notations of above and below substates from [13]:

Definition 6. The set of above substates coded by the node \underline{n} is

$$\begin{aligned} \mathcal{A}(\underline{n}) &\subseteq \{\langle x[j+1], x[j+2], \dots, x[J-1] \rangle \in D^{(j+1)} \times D^{(j+2)} \times \dots \times D^{(J-1)}\}, \\ \mathbf{x} \in \mathcal{A}(\underline{n}) &\iff \text{child}(\text{child}(\dots \text{child}(\underline{r}, x[J-1]) \dots, x[j+2]), x[j+1]) = \underline{n}, \end{aligned}$$

where $j = \text{level}(\underline{n}) - 1$, i.e. $\mathcal{A}(\underline{n})$ is the set of all paths in the MDD from \underline{r} to \underline{n} .

Definition 7. The set of below substates coded by the node \underline{n} is

$$\begin{aligned} \mathcal{B}(\underline{n}) &\subseteq \{\langle x[0], x[1], \dots, x[j] \rangle \in D^{(0)} \times D^{(1)} \times \dots \times D^{(j)}\}, \\ \mathbf{x} \in \mathcal{B}(\underline{n}) &\iff \text{child}(\text{child}(\dots \text{child}(\underline{n}, x[j]) \dots, x[1]), x[0]) = \underline{1}, \end{aligned}$$

where $j = \text{level}(\underline{n}) - 1$, i.e. $\mathcal{B}(\underline{n})$ is the set of all paths in the MDD from \underline{n} to $\underline{1}$.

Proposition 1. If \underline{n} and \underline{m} are distinct nonterminal nodes of a quasi-reduced ordered MDD, $\mathcal{A}(\underline{n}) \cap \mathcal{A}(\underline{m}) = \emptyset$ and $\mathcal{B}(\underline{n}) \neq \mathcal{B}(\underline{m})$.

Proof. We prove the statements by contradiction. Let $\mathbf{a} \in \mathcal{A}(\underline{n}) \cap \mathcal{A}(\underline{m})$. If we follow the path \mathbf{a} from \underline{r} we arrive at \underline{n} because $\mathbf{a} \in \mathcal{A}(\underline{n})$. However, we also arrive at \underline{m} , because $\mathbf{a} \in \mathcal{A}(\underline{m})$. Since $\underline{n} \neq \underline{m}$, such \mathbf{a} cannot exist. $\mathcal{A}(\underline{n})$ and $\mathcal{A}(\underline{m})$ must be disjoint.

Now suppose that there are $\underline{n}, \underline{m} \in V_N$ such that $\mathcal{B}(\underline{n}) = \mathcal{B}(\underline{m})$. Because the set of paths $\mathcal{B}(\underline{n})$ fully describes the subgraph reachable from \underline{n} , this means the subgraphs reachable from \underline{n} and \underline{m} are isomorphic. This is impossible since the MDD is quasi-reduced. $\mathcal{B}(\underline{n})$ and $\mathcal{B}(\underline{m})$ must be distinct. \square

Proposition 2. *The environment of the local state $x^{(j)}$, defined in Eq. (10), may be written as*

$$\text{env } x^{(j)} = \{\langle \mathbf{b}, \mathbf{a} \rangle : \underline{n} \in V_{j+1}, \mathbf{a} \in \mathcal{A}(\underline{n}), \mathbf{b} \in \mathcal{B}(\text{child}(\underline{n}, x^{(j)}))\}.$$

Proof. Any reachable state $\langle \mathbf{z} \rangle \in \text{RS}$ that has $z[j] = x$ is represented by a path from \underline{r} to $\underline{1}$ in the MDD that passes through a pair of nodes $\underline{n} \in V_{j+1}$ and $\underline{k} = \text{child}(\underline{n}, x^{(j)})$. Therefore, some path $\mathbf{a} \in \mathcal{A}(\underline{n})$ must be followed from \underline{r} to reach \underline{n} , then after traversing the edge between \underline{n} and \underline{k} , some path $\mathbf{b} \in \mathcal{B}(\underline{k})$ must be followed from \underline{k} to $\underline{1}$.

This means all paths from \underline{r} to $\underline{1}$ containing $x^{(j)}$ are of the form $\mathbf{z} = (\mathbf{b}, x, \mathbf{a})$ and the converse also holds. Thus, $\hat{\mathbf{z}}^{(j)} = \langle \mathbf{a}, \mathbf{b} \rangle$ holds for some \mathbf{a} and \mathbf{b} defined as above for all reachable states \mathbf{z} . \square

The relation $\sim^{(j)}$ over $\text{RS}^{(j)}$ can be expressed with $\mathcal{A}(\underline{n})$ and $\mathcal{B}(\underline{n})$ in a way that can be handled with symbolic techniques.

Proposition 3. *The relation $x^{(j)} \sim^{(j)} y^{(j)}$ can be formulated as*

$$x^{(j)} \sim^{(j)} y^{(j)} \iff \text{edges}(x, j) = \text{edges}(y, j),$$

where $\text{edges}(z, j) = \{\langle \underline{n}, \text{child}(\underline{n}, z) \rangle : \underline{n} \in V_{j+1}\}.$

Proof. Recall that $x^{(j)} \sim^{(j)} y^{(j)}$ is defined as $\text{env } x^{(j)} = \text{env } y^{(j)}$ in Eq. (11). Let X and Y be the environments of $x^{(j)}$ and $y^{(j)}$, respectively, so that $x^{(j)} \sim^{(j)} y^{(j)}$ holds if and only if $X = Y$. Define

$$X(\underline{n}) = \{\mathbf{b} : \langle \mathbf{b}, \mathbf{a} \rangle \in X, \mathbf{a} \in \mathcal{A}(\underline{n})\}, \quad Y(\underline{n}) = \{\mathbf{b} : \langle \mathbf{b}, \mathbf{a} \rangle \in Y, \mathbf{a} \in \mathcal{A}(\underline{n})\}.$$

Observe that $X = Y$ if and only if $X(\underline{n}) = Y(\underline{n})$ for all $\underline{n} \in V_{j+1}$, because the sets $\{X(\underline{n}) \times \mathcal{A}(\underline{n})\}_{\underline{n} \in V_{j+1}}$ and $\{Y(\underline{n}) \times \mathcal{A}(\underline{n})\}_{\underline{n} \in V_{j+1}}$ are partitions of X and Y .

According to Proposition 2,

$$X(\underline{n}) = \mathcal{B}(\text{child}(\underline{n}, x)), \quad Y(\underline{n}) = \mathcal{B}(\text{child}(\underline{n}, y)).$$

Thus, $X(\underline{n}) = Y(\underline{n})$ if and only if $\text{child}(\underline{n}, x) = \text{child}(\underline{n}, y)$, because the \mathcal{B} -sets are distinct for each node. Hence $X(\underline{n}) = Y(\underline{n})$ for all $\underline{n} \in V_{j+1}$ is equivalent to the statement $\text{edges}(x, j) = \text{edges}(y, j)$. \square

Proposition 3 can be interpreted as the statement that $x^{(j)} \sim^{(j)} y^{(j)}$ if and only if the MDD edges corresponding to $x^{(j)}$ are always parallel, i.e. from the node \underline{n} they all go to the same node $\underline{m}(\underline{n})$, which only depends on \underline{n} , for all $\underline{n} \in V_{j+1}$.

4.3 Symbolic State Indexing

The last step of the stochastic analysis workflow is the calculation of engineering measures by weighted sums of state probabilities, which were obtained by

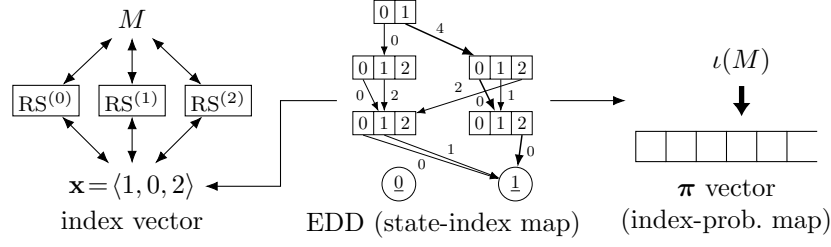


Fig. 5. Index manipulations for EDD-based state spaces.

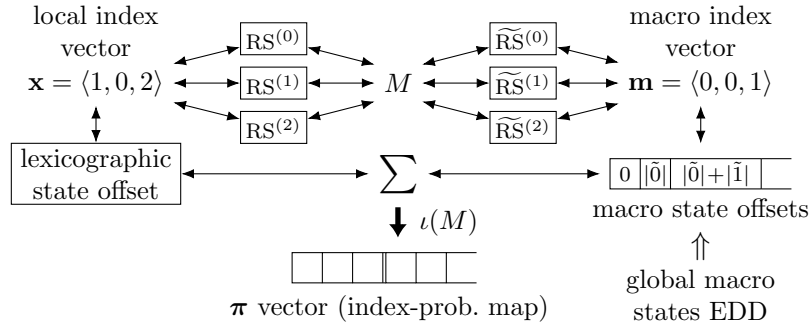


Fig. 6. Index manipulations for EDD-based macro state spaces.

the numerical solution algorithm. This requires iteration over all states and the corresponding indices of the probability vector.

In this section, we review techniques for state-index mapping in superposed stochastic Petri nets. An implementation of the mapping is presented for macro state decompositions in symbolic form.

The EDD-based state-index mapping is depicted in Fig. 5. A vector of local macro state indices is formed by observing the local states of the components. Both forward and reverse mapping between linear indices $\iota(M)$ of the probability vector and index vectors \mathbf{x} is possible [12].

In block Kronecker decompositions, the generator matrix, as well as the vector π is partitioned according to the global macro states \tilde{RS} . Within a partition, markings are ordered lexicographically by their local state indices. Although it is possible to construct an EDD representing this ordering, it can easily grow large, due to the intertwined ordering.

To efficiently store the index mapping (Fig. 6), a macro state offset vector \mathbf{o} of length \tilde{n} is populated with the starting offsets of the global states by enumeration of the global macro state space. The i th marking in the lexicographic ordering within the global macro state \tilde{m} has the linear index $x = o[m] + i$. Conversion between linear macro state indices m and macro state index vectors \mathbf{m} is performed by an EDD, analogously to the mapping in Fig. 5.

5 Evaluation

The decomposition algorithms were implemented in the **PetriDotNet** modelling framework, which supports structural analysis of Petri nets, saturation-based CTL and LTL model checking and reachability checking as well as Markovian stochastic analysis. The algorithms may be used in Markovian transient, sensitivity and time-to-first-failure analysis in addition to steady state analysis.

Our symbolic block Kroncker decomposition approach is compared with sparse generator matrices and the explicit block Kronecker decomposition algorithm by Buchholz [7]. The explicit decomposition algorithm was executed on an explicit representation of the state space as a hash table, while symbolic algorithms used saturation and MDDs. Steady-state analysis was performed using the BiCGSTAB [24] numerical linear equation solver with a tolerance of 10^{-10} .

Running time was limited to 1 hour on virtual machines with 8 execution threads and 30 GiB memory.

Three scalable families of models were used in the evaluation:

- The *SR* family of models are upscaled variants of the *SharedResource* model that served as a running example throughout this paper. The model was extended with additional clients, moreover, the number of tokens was increased. In *SR-Sym*, each transition rate is set to 1.0, while in *SR-Asym*, different rates were chosen for the transitions.
- The *KanBan* SPN models from [11] describe the kanban manufacturing system with various resource pool sizes.
- Members of the *Cloud* family are SPN performability models of a cloud architecture [17]. Scaling of the state space is achieved by changing the number of physical and virtual machines. Some aspects of the models from [17] were modified, as our tool currently does not support the GPSN formalism.

The models in several SPN formats and the **PetriDotNet** 1.5 tool are available at our website¹.

Table 1 shows the measured execution times and memory usages. The execution time of the decomposition algorithm t_{macro} , the construction of the generator matrix t_{gen} and the BiCGSTAB solver t_{solve} is displayed, as well as the peak memory usage M_{peak} . Minimum values of every measure are emphasised in bold for each model.

The symbolic decomposition algorithm was executed under 100ms for all the studied models, even in cases when the BiCGSTAB algorithm timed out. Block Kronecker based analysis with symbolic state space also consumed the least amount of memory, except in a single variation of the *KanBan* model.

Matrix construction was also fastest with symbolic decomposition. While the same matrices are constructed after explicit decomposition, the matrices were built slower based on the explicit state space than on MDDs due to fast access to the decision diagram data structures.

¹ <https://inf.mit.bme.hu/en/petridotnet/stochasticanalysis>

Table 1. Measurement results.

Model	States	Sparse			Explicit BK				Symbolic BK			
		t_{gen}	t_{solve}	M_{peak}	t_{macro}	t_{gen}	t_{solve}	M_{peak}	t_{macro}	t_{gen}	t_{solve}	M_{peak}
SR-Sym	26 464	147 ms	296 ms	40 M	464 ms	562 ms	303 ms	55 M	23 ms	142 ms	370 ms	38 M
	842 051	2 s	10 s	296 M	22 s	11 s	10 s	1035 M	23 ms	582 ms	12 s	98 M
	$1.1 \cdot 10^7$	24 s	182 s	3404 M	530 s	183 s	231 s	12 G	24 ms	5 s	155 s	791 M
	$8.2 \cdot 10^7$	215 s	1985 s	25 G	Timed out				26 ms	31 s	1612 s	5563 M
SR-Asym	26 464	132 ms	482 ms	42 M	455 ms	402 ms	580 ms	57 M	26 ms	125 ms	638 ms	39 M
	842 051	2 s	17 s	296 M	22 s	10 s	20 s	1035 M	23 ms	519 ms	21 s	99 M
	$1.1 \cdot 10^7$	22 s	327 s	3404 M	540 s	246 s	463 s	12 G	27 ms	6 s	285 s	793 M
	$8.2 \cdot 10^7$	217 s	Timed out		Timed out				31 ms	32 s	Breakdown	
KanBan	58 400	181 ms	1 s	51 M	250 ms	847 ms	3 s	77 M	24 ms	130 ms	2 s	45 M
	454 475	1 s	11 s	181 M	1 s	4 s	31 s	448 M	26 ms	285 ms	19 s	171 M
	$2.5 \cdot 10^6$	5 s	81 s	388 M	6 s	28 s	240 s	2276 M	22 ms	1 s	124 s	780 M
	$1.1 \cdot 10^7$	33 s	436 s	3727 M	31 s	138 s	1274 s	10 G	25 ms	5 s	737 s	3591 M
Cloud	$2.7 \cdot 10^6$	38 s	446 s	5846 M	119 s	48 s	913 s	3361 M	64 ms	4 s	278 s	291 M
	$2.0 \cdot 10^7$	Out of memory			2021 s	464 s	Timed out		29 ms	9 s	Timed out	
	$1.3 \cdot 10^8$	Out of memory				Timed out			32 ms	75 s	Timed out	
	$8.2 \cdot 10^8$	Out of memory				Timed out			42 ms	574 s	Out of memory	

The state space data structure must be kept in memory throughout the numerical solution, because it is needed in the phase of engineering measure evaluation. Thus, BiCGSTAB ran slower after explicit decomposition due to the pressure on the garbage collector (GC) caused by the larger size of the state space data structures. For smaller models, numerical solution with sparse matrices was faster than with block Kronecker matrices. However, for larger models the decomposed matrix utilized the memory bandwidth and caches more efficiently.

The sole difference between the *Sym* and *Asym* versions of the *SR* models are the transition rates, which only affected the numerical solution time. The structure of the state space remained identical.

Failures of BiCGSTAB included exhaustion of the time limit and the available memory. In the largest *SR-Asym* model, a numerical breakdown condition occurred. This condition may be handled by switching to a more stable solver, such as Jacobi or Gauss–Seidel iteration [23, Section 2.2]. These solvers are also implemented in the PetriDotNet framework.

6 Conclusion and Future Work

In this paper we introduced an efficient stochastic analysis approach using symbolic algorithms to explore the possible behaviours of the system and decomposition-based stochastic analysis algorithms to efficiently compute stationary measures of stochastic Petri nets. In our work we established an efficient mapping technique which can bridge the gap between the encoded state space representation and the decomposition-based numerical algorithms. This algorithm supports the analysis of Petri net models with huge state spaces and complex behaviours. Measurements on models with various sizes and character-

istics showed the effectiveness of the introduced approach and the benefits of the new mapping algorithm.

In the future, we plan to further extend our stochastic analysis framework with other numerical solution algorithms such as the SPLIT algorithm for Kronecker products. In addition, we will also investigate GPU and distributed implementation of the available algorithms.

Acknowledgement. This work was partially supported by the ARTEMIS JU and the Hungarian National Research, Development and Innovation Fund in the frame of the R5-COP project. This research was partially performed within the framework of the grant of the Hungarian Scientific Research Fund (grant no. OTKA K101150).

References

1. Ajmone Marsan, M.: Stochastic Petri nets: an elementary introduction. In: Rozenberg, G. (ed.) *Advances in Petri Nets 1989*, covers the 9th European Workshop on Applications and Theory in Petri Nets, held in Venice, Italy in June 1988, selected papers. *Lecture Notes in Computer Science*, vol. 424, pp. 1–29. Springer (1988)
2. Baier, C., Katoen, J.P., Hermanns, H.: Approximative symbolic model checking of continuous-time Markov chains. In: *CONCUR'99 Concurrency Theory*, pp. 146–161. Springer (1999)
3. Barrett, R., Berry, M.W., Chan, T.F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., Van der Vorst, H.: *Templates for the solution of linear systems: building blocks for iterative methods*, vol. 43. Siam (1994)
4. Bause, F., Buchholz, P., Kemper, P.: A toolbox for functional and quantitative analysis of DEVS. In: Puigjaner, R., Savino, N.N., Serra, B. (eds.) *Computer Performance Evaluation: Modelling Techniques and Tools*, 10th International Conference, Tools '98, Palma de Mallorca, Spain, September 14–18, 1998, *Proceedings. Lecture Notes in Computer Science*, vol. 1469, pp. 356–359. Springer (1998)
5. Benoit, A., Plateau, B., Stewart, W.J.: Memory-efficient Kronecker algorithms with applications to the modelling of parallel systems. *Future Generation Comp. Syst.* 22(7), 838–847 (2006)
6. Bianco, A., De Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: *Foundations of Software Technology and Theoretical Computer Science*. pp. 499–513. Springer (1995)
7. Buchholz, P.: Hierarchical structuring of superposed GSPNs. *IEEE Trans. Software Eng.* 25(2), 166–181 (1999)
8. Buchholz, P., Ciardo, G., Donatelli, S., Kemper, P.: Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models. *INFORMS Journal on Computing* 12(3), 203–222 (2000)
9. Buchholz, P., Kemper, P.: On generating a hierarchy for GSPN analysis. *SIGMETRICS Performance Evaluation Review* 26(2), 5–14 (1998)
10. Buchholz, P., Kemper, P.: Kronecker based matrix representations for large Markov models. In: *Validation of Stochastic Systems*, pp. 256–295. Springer (2004)
11. Ciardo, G., Jones, R.L., Miner, A.S., Siminiceanu, R.: Logical and stochastic modeling with SMART. In: *Computer Performance Evaluation. Modelling Techniques and Tools*, pp. 78–97. Springer (2003)

12. Ciardo, G., Miner, A.S.: A data structure for the efficient Kronecker solution of GSPNs. In: *Petri Nets and Performance Models*, 1999. Proceedings. The 8th International Workshop on. pp. 22–31. IEEE (1999)
13. Ciardo, G., Zhao, Y., Jin, X.: Ten years of saturation: A petri net perspective. *T. Petri Nets and Other Models of Concurrency* 5, 51–95 (2012)
14. Czekster, R.M., Rose, C.A.F.D., Fernandes, P.H.L., de Lima, A.M., Webber, T.: Kronecker descriptor partitioning for parallel algorithms. In: McGraw, R.M., Im-sand, E.S., Chinni, M.J. (eds.) *Proceedings of the 2010 Spring Simulation Multiconference, SpringSim 2010, Orlando, Florida, USA, April 11-15, 2010*. p. 242. SCS/ACM (2010)
15. Donatelli, S.: Superposed generalized stochastic Petri nets: Definition and efficient solution. In: Valette, R. (ed.) *Application and Theory of Petri Nets 1994*, 15th International Conference, Zaragoza, Spain, June 20-24, 1994, Proceedings. Lecture Notes in Computer Science, vol. 815, pp. 258–277. Springer (1994)
16. Fernandes, P., Presotto, R., Sales, A., Webber, T.: An alternative algorithm to multiply a vector by a Kronecker represented descriptor. In: *21st UK Performance Engineering Workshop*. pp. 57–67 (2005)
17. Ghosh, R.: Scalable stochastic models for cloud services. Ph.D. thesis, Duke University (2012)
18. Kemper, P.: Numerical analysis of superposed GSPNs. *IEEE Trans. Software Eng.* 22(9), 615–628 (1996)
19. Kwiatkowska, M., Mehmood, R., Norman, G., Parker, D.: A symbolic out-of-core solution method for Markov models. *Electronic Notes in Theoretical Computer Science* 68(4), 589–604 (2002)
20. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal on Software Tools for Technology Transfer* 6(2), 128–142 (2004)
21. Miner, A.S.: Efficient solution of GSPNs using canonical matrix diagrams. In: *Petri Nets and Performance Models*, 2001. Proceedings. 9th International Workshop on. pp. 101–110. IEEE (2001)
22. Roux, P., Siminiceanu, R.: Model checking with edge-valued decision diagrams. In: Muñoz, C.A. (ed.) *Second NASA Formal Methods Symposium - NFM 2010*, Washington D.C., USA, April 13-15, 2010. Proceedings. NASA Conference Proceedings, vol. NASA/CP-2010-216215, pp. 222–226 (2010)
23. Stewart, W.J.: *Probability, Markov chains, queues, and simulation: the mathematical basis of performance modeling*. Princeton University Press (2009)
24. Van der Vorst, H.A.: Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal on scientific and Statistical Computing* 13(2), 631–644 (1992)
25. Wan, M., Ciardo, G., Miner, A.S.: Approximate steady-state analysis of large Markov models based on the structure of their decision diagram encoding. *Performance Evaluation* 68(5), 463–486 (2011)
26. Zhao, Y., Ciardo, G.: A two-phase Gauss-Seidel algorithm for the stationary solution of EVMDD-encoded CTMCs. In: *Ninth International Conference on Quantitative Evaluation of Systems, QEST 2012, London, United Kingdom, September 17-20, 2012*. pp. 74–83. IEEE Computer Society (2012)